Natural Language Processing

2: Tokenization and Segmentation

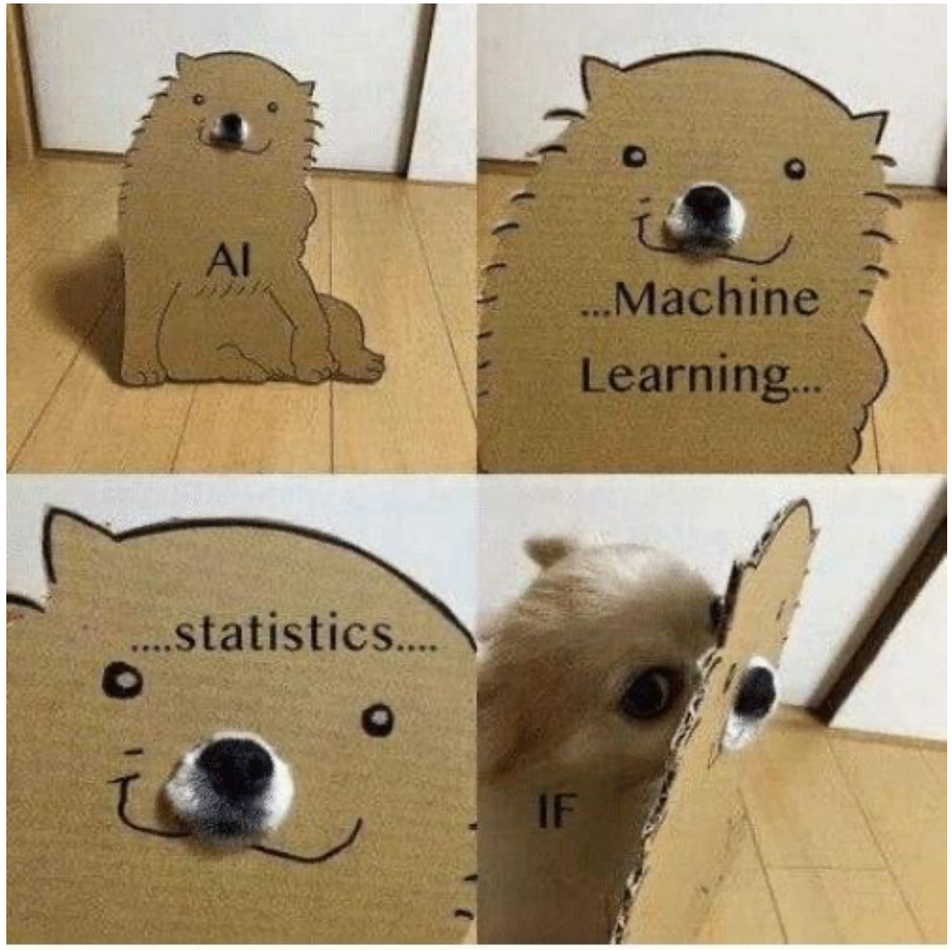
Objectives

to be able to answer questions

(1) What methods exist for Tokenization and Segmentation?

(2) What is the place of a tokenizer in NLP pipeline?

(3) How does BPE work?



IF IF IF IF IF IF IF IF IF WE!

Words and Corpora

How many words in a sentence?

(1)"I do uh main- mainly business data processing"

- Fragments, filled pauses

(2)"Seuss's cat in the hat is different from other cats!"

- Lemma: same stem, part of speech, rough word sense
  - **cat** and **cats** = same lemma
- Wordform: the full inflected surface form
  - **cat** and **cats** = different wordforms

How many words in a sentence?

They lay back on the San Francisco grass and looked at the stars and their

- Type: an element of the vocabulary
- Token: an instance of that type in running text
- How many?
  - 15 tokens (or 14)
  - 13 types (or 12) (or 11?)

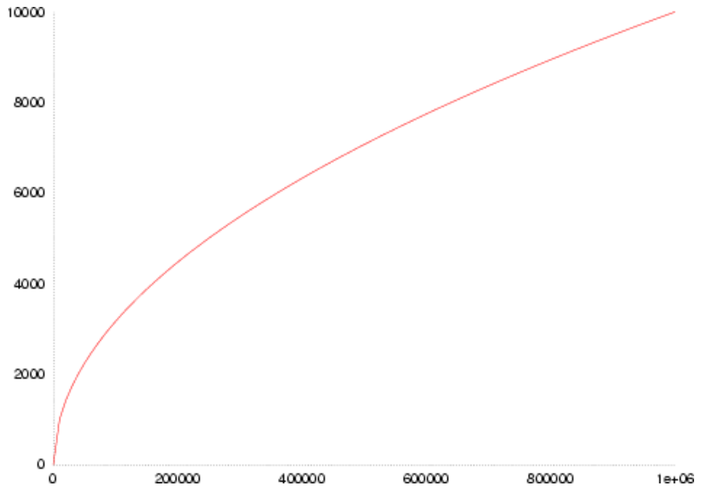
How many words in a corpus?

N = number of tokens

V = vocabulary = set of types, **|*V*|** is size of vocabulary

Heaps Law = Herdan's Law : $|V| = kN^{\beta}$ where often $.4 < \beta < .75$

i.e., vocabulary size grows with > square root of the number of word tokens



Corpus	Tokens = N	Types = |V|
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
COCA	440 million	2 million
Google N-grams	1 trillion	13+ million
Common Crawl / mC4	6.3 trillion	??

Corpora

corpora = corpuses

Words don't appear out of nowhere!

A text is produced by

(1) a specific writer(s),

(2) at a specific time,

(3) in a specific variety,

(4) of a specific language,

(5) for a specific function.

Corpora vary along dimension

- Language: 7097 languages in the world
- Variety: like African American Language varieties.
  - AAE Twitter posts might include forms like "iont" (I don't)
- Code switching, e.g., Spanish/English, Hindi/English:

  S/E: Por primera vez veo a @username actually being hateful! It was beautiful:)
  [For the first time I get to see @username actually being hateful! it was beautiful:)]

  H/E: dost tha or ra- hega ... dont wory ... but dherya rakhe
  [“he was and will remain a friend ... don’t worry ... but have faith”]

- Genre: newswire, fiction, scientific articles, Wikipedia
- Author Demographics: writer's age, gender, ethnicity

Corpus datasheets

Motivation:

(1) Why was the corpus collected?
(2) By whom?
(3) Who funded it?

Situation: In what situation was the text written?

Collection process: If it is a subsample how was it sampled? Was there consent? Pre-processing

Annotation process, language variety, demographics, etc.

Example: NEREL, NEREL-BIO

(1) Largest Russian Dataset of Named Entities, Relations and Entity links https://aclanthology.org/2021.ranlp-1.100.pdf

(2) NEREL-BIO: https://paperswithcode.com/dataset/nerel-bio

	Dataset	Lang	#NE inst. (Types)	Max Depth	#Rel inst. (Types)
1	CoNLL03 (Tjong Kim Sang and De Meulder, 2003)	en	34.5K (4)	1	–
	Ontonotes (Hovy et al., 2006)	en	104K (19)	1	–
2	ACE2005 (Walker et al., 2006)	en	30K (7)	6	8.3K(6)
	NNE (Ringland et al., 2019)	en	279K (114)	6	–
	No-Sta-D (Benikova et al., 2014)	de	41K (12)	2	–
	Digitoday (Ruokolainen et al., 2019)	fi	19K (6)	2	–
	DAN+ (Plank et al., 2020)	da	6.4K (4)	2	–
3	TACRED (Zhang et al., 2017)	en	(3)	1	22.8K (42)
	DocRED (Yao et al., 2019)	en	132K (6)	1	56K (96)
4	Gareev (Gareev et al., 2013)	ru	44K (2)	1	–
	Collection3 (Mozharova and Loukachevitch, 2016)	ru	26.4K(3)	1	–
	FactRuEval (Starostin et al., 2016)	ru	12K (3)	2	1K (4)
	BSNLP (Piskorski et al., 2019)	ru	9K (5)	1	–
	RuREBUS (Ivanin et al., 2020)	ru	121K (5)	1	14.6K (8)
	RURED (Gordeev et al., 2020)	ru	22.6K (28)	1	5.3K(34)
	NEREL (ours)	ru	56K (29)	6	39K (49)

Datasets vs. Corpora vs. Text Collections

(1) Discussion: What is the difference?

Tokenization

Tokenization

recap

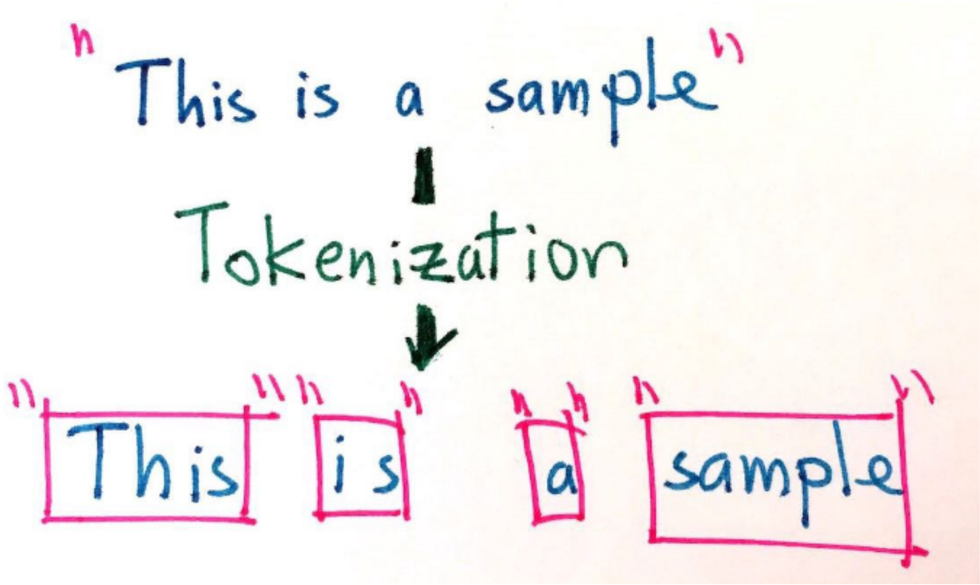
(1) Breaking the input text into meaningful elements, smaller pieces or "tokens".

(2) Tokens can be:

  (1) elements for further semantic processing

  (2) word, sentence, paragraph

(3) Certain characters, such as punctuation marks, can be removed at the same time.



Regular Expressions: Recap

Regular expressions

(1)A formal language for specifying text strings

(2)How can we search for any of these?

- (1)woodchuck
- (2)woodchucks
- (3)Woodchuck
- (4)Woodchucks

Regular Expressions: Disjunctions

(1)Letters inside square brackets []

Pattern	Matches
`[wW]oodchuck`	Woodchuck, woodchuck
`[1234567890]`	Any digit

(3)Ranges `[A-Z]`

Pattern	Matches	
`[A-Z]`	An upper case	`Drenched Blossoms`
`[a-z]`	A lower case letter	`my beans were impatient`
`[0-9]`	A single digit	`Chapter 1: Down the Rabbit Hole`

Regular Expressions: Negation in Disjunction

(1) Negations `[^Ss]`

(1) Carat means negation only when first in []

Pattern	Matches	
`[^A-Z]`	Not an upper case letter	`Oyfn pripetchik`
`[^Ss]`	Neither ‘S’ nor ‘s’	`I have no exquisite reason”`
`[^e^]`	Neither e nor ^	`Look here`
`a^b`	The pattern a carat	`Look up a^b now`

Regular Expressions: More Disjunction

(1)Woodchuck is another name for groundhog!

(2)The pipe | for disjunction

Pattern	Matches
`groundhog|woodchuck`	woodchuck
`yours|mine`	yours
`a|b|c`	= [abc]
`[gG]roundhog|[Ww]oodchuck`	Woodchuck

Regular Expressions: ? *+ .

Pattern	Matches	
`colou?r`	Optional previous char	color colour
`oo*h!`	0 or more of previous char	oh! ooh! oooh! ooooh!
`o+h!`	1 or more of previous char	oh! ooh! oooh! ooooh!
`baa+`		baa baaa baaaa
`beg.n`		begin begun begun

Regular Expressions: Anchors ^ $

Pattern	Matches
^[A-Z]	Palo Alto
^[^A-Za-z]	1 "Hello"
\.$	The end.
.$	The end? The end!

Example

Find me all instances of the word “the” in a text

```
the
```

Misses capitalized examples

```
[tT]he
```

Incorrectly returns `other` or `theology`

```
[^a-zA-Z][tT]he[^a-zA-Z]
```

Errors

The process we just went through was based on fixing two kinds of errors:

1. Matching strings that we should not have matched (there, then, other)

False positives (Type I errors)

2. Not matching things that we should have matched (The)

False negatives (Type II errors)

Errors cont.

(1) In NLP we are always dealing with these kinds of errors.

(2) Reducing the error rate for an application often involves two antagonistic efforts:

(1) Increasing accuracy or precision
(minimizing false positives)

(2) Increasing coverage or recall
(minimizing false negatives).

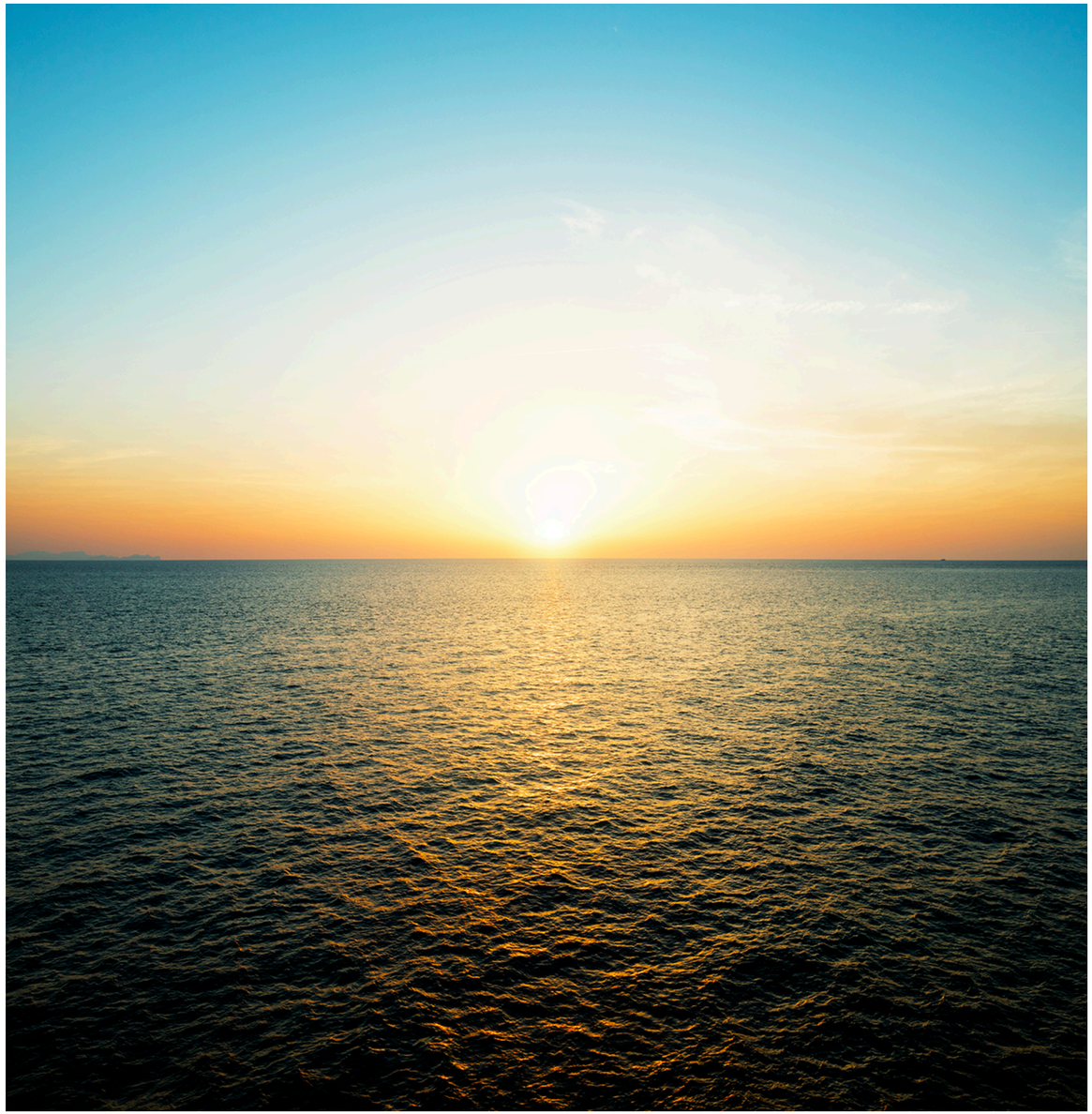
Summary

Regular expressions play a surprisingly large role

- Sophisticated sequences of regular expressions are often the first model for any text processing text

For hard tasks, we use machine learning classifiers

- But regular expressions are still used for pre-processing, or as features in the classifiers
- Can be very useful in capturing generalizations



Break, 1 week

Text Normalization

Every NLP task requires text normalization:

1. Tokenizing (segmenting) words
2. Normalizing word formats
3. Segmenting sentences

Space-based tokenization

(1) A very simple way to tokenize
    (1) For languages that use space characters between words
    (2) Segment off a token between instances of spaces
(2) Unix tools for space-based tokenization
    (1) The "tr" command
    (2) Inspired by Ken Church's UNIX for Poets
    (3) Given a text file, output the word tokens and their frequencies

Simple Tokenization in UNIX

(1)(Inspired by Ken Church's UNIX for Poets.)

(2)Given a text file, output the word tokens and their frequencies

```
tr -sc 'A-Za-z' '\n' < shakes.txt | sort | uniq –c
```

```
1945 A
  72 AARON
  19 ABBESS
   5 ABBOT
 ... ...
```

```
25 Aaron
 6 Abate
 1 Abates
 5 Abbess
 6 Abbey
 3 Abbot
... …
```

The first step: tokenizing

```
tr -sc 'A-Za-z' '\n' < shakes.txt | head
```

```
THE
SONNETS
by
William
Shakespeare
From
fairest
creatures
We
...
```

The second step: sorting

```
tr -sc 'A-Za-z' '\n' < shakes.txt | sort | head

A
A
A
A
A
A
A
A
A
A
...
```

More counting

(1) Merging upper and lower case

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c
```

(2) Sorting the counts

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c | sort -n -r
```

```
  23243 the
  22225 i
  18618 and
  16339 to
  15687 of
  12780 a
  12163 you
  10839 my
  10005 in
   8954 d
```

What happened here?

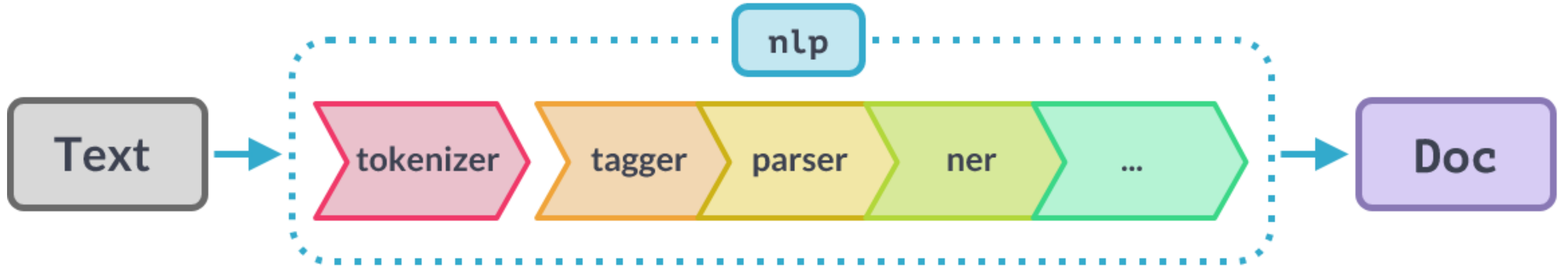
Issues in Tokenization

(1)Can't just blindly remove punctuation:
    (1)m.p.h., Ph.D., AT&T, cap'n
    (2)prices ($45.55)
    (3)dates (01/02/06)
    (4)URLs (http://www.stanford.edu)
    (5)hashtags (#nlproc)
    (6)email addresses (someone@cs.colorado.edu)

(2)Clitic: a word that doesn't stand on its own
    (1)"are" in we're, French "je" in j'ai, "le" in l'honneur

(3)When should multiword expressions (MWE) be words?
    (1)New York, rock 'n' roll

Tokenization in Spacy




```
import spacy
nlp = spacy.load("en_core_web_md")
doc = nlp("We are learning NLP using spaCy.")
print ([token.text for token in doc])
```

✓ 2.6s

```
['We', 'are', 'learning', 'NLP', 'using', 'spaCy', '.']
```

https://medium.com/analytics-vidhya/nlp-with-spacy-tutorial-part-2-tokenization-and-sentence-segmentation-352df790a214

https://machinelearningknowledge.ai/complete-guide-to-spacy-tokenizer-with-examples/

Tokenization in languages without spaces

Many languages (like Chinese, Japanese, Thai) don't use spaces to separate words!

How do we decide where the token boundaries should be?

Word tokenization in Chinese

Chinese words are composed of characters called "**hanzi**" (or sometimes just "**zi**")

Each one represents a meaning unit called a morpheme.

Each word has on average 2.4 of them.

But deciding what counts as a word is complex and not agreed upon.

李叶真的跟她的妈妈不一样，她看起来又隘又屌，还经常生病。她总是喜欢生气，生气的时候总是媸。如果李叶的妈妈听到她媸，就会很生气。所有的人都不喜欢这个孩子，他们从来没有让过这样的孩子。为了不让李叶媸，她的阿姨总是很听李叶的话。李叶资弦什么，她的阿姨就朐她什么。李叶觉得在这个家里只有她的阿姨关心她。

How to do word tokenization in Chinese?

the table down there

姚明进入总决赛 “Yao Ming reaches the finals”

3 words?
姚明 进入 总决赛
YaoMing reaches finals

5 words?
姚 明 进入 总 决赛
Yao Ming reaches overall finals

7 characters? (don't use words at all):
姚 明 进 入 总 决 赛
Yao Ming enter enter overall decision game

Word tokenization / segmentation

In Chinese it's common to just treat each character (zi) as a token.

(1)So the **segmentation** step is very simple

In other languages (like Thai and Japanese), more complex word segmentation is required.

(2)The standard algorithms are neural sequence models trained by supervised machine learning.

Break, 5 min.

Byte Pair Encoding

Another option for text tokenization

Instead of

(1)white-space segmentation

(2)single-character segmentation

Use the data to tell us how to tokenize.

Subword tokenization (because tokens can be parts of words as well as whole words)

Subword tokenization

(1)Four common algorithms:

(1)Byte-Pair Encoding (BPE) (Sennrich et al., 2016)

(2)Unigram language modeling tokenization (Kudo, 2018)

(3)WordPiece (Schuster and Nakajima, 2012)

(4)SentencePiece (Kudo and Richardson, 2018) https://arxiv.org/abs/1808.06226

(2)All have 2 parts:

(1)A token **learner** that takes a raw training corpus and induces a vocabulary (a set of tokens).

(2)A token **segmenter** that takes a raw test sentence and tokenizes it according to that vocabulary

Byte Pair Encoding (BPE) token learner

Let vocabulary be the set of all individual characters

= {A, B, C, D,..., a, b, c, d....}

(1) Repeat:

  (1) Choose the two symbols that are most frequently adjacent in the training corpus (say 'A', 'B')

  (2) Add a new merged symbol 'AB' to the vocabulary

  (3) Replace every adjacent 'A' 'B' in the corpus with 'AB'.

(2) Until *k* merges have been done.

BPE token learner algorithm

(1) We shall consider the following text:

low low low low low lowest lowest newer newer newer newer newer newer wider wider wider new new

Algorithm 1 Learn BPE operations

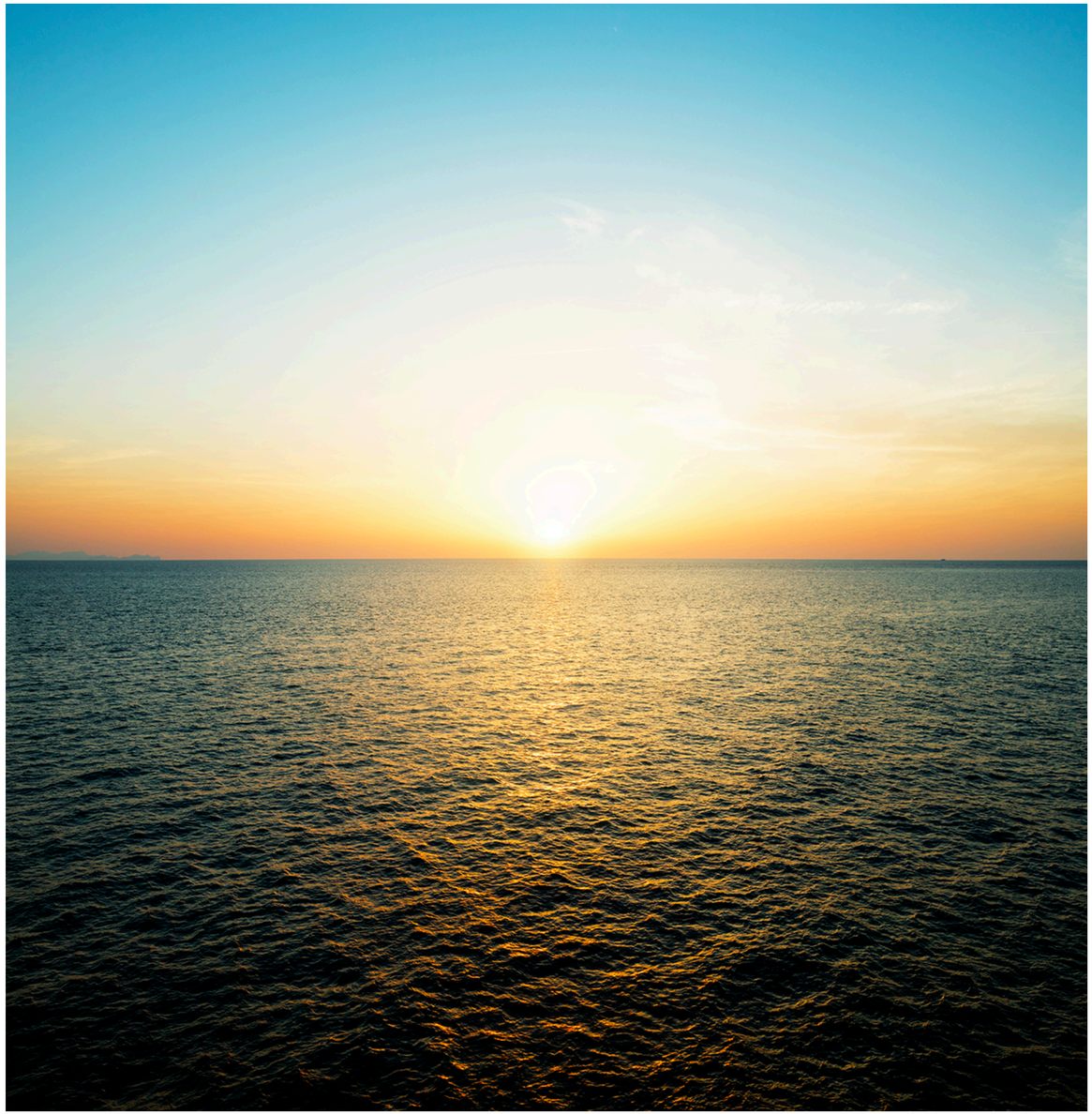
```
import re, collections

def get_stats(vocab):
  pairs = collections.defaultdict(int)
  for word, freq in vocab.items():
    symbols = word.split()
    for i in range(len(symbols)-1):
      pairs[symbols[i],symbols[i+1]] += freq
  return pairs

def merge_vocab(pair, v_in):
  v_out = {}
  bigram = re.escape(' '.join(pair))
  p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
  for word in v_in:
    w_out = p.sub(''.join(pair), word)
    v_out[w_out] = v_in[word]
  return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
  pairs = get_stats(vocab)
  best = max(pairs, key=pairs.get)
  vocab = merge_vocab(best, vocab)
  print(best)
```

Byte Pair Encoding (BPE)

Most subword algorithms are run inside space-separated tokens.

So we commonly first add a special end-of-word symbol '__' before space in training corpus

Next, separate words into letters.



BPE token learner

Original (very fascinating🙄) corpus:

low low low low low lowest lowest newer newer newer newer newer newer wider wider wider new new

Add end-of-word tokens, resulting in this vocabulary:

corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w
2	l o w e s t _	
6	n e w e r _	
3	w i d e r _	
2	n e w _	

BPE: 1st Merge operation

corpus	vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w
2 l o w e s t _	
6 n e w e r _	
3 w i d e r _	
2 n e w _	

Merge e r to er

corpus	vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w, er
2 l o w e s t _	
6 n e w er _	
3 w i d er _	
2 n e w _	

BPE: 2nd Merge operation

corpus	vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w, er
2 l o w e s t _	
6 n e w er _	
3 w i d er _	
2 n e w _	

Merge er _ to er_

corpus	vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_
2 l o w e s t _	
6 n e w er_	
3 w i d er_	
2 n e w _	

BPE: 3rd Merge operation

corpus	vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_
2 l o w e s t _	
6 n e w er_	
3 w i d er_	
2 n e w _	

Merge n e to ne

corpus	vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne
2 l o w e s t _	
6 ne w er_	
3 w i d er_	
2 ne w _	

BPE

(1) The next merges are:

Merge	Current Vocabulary
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_

BPE token segmenter algorithm

On the test data, run each merge learned from the training data:

(1) Greedily

(2) In the order we learned them
(test frequencies don't play a role)

So: merge every e r to er, then merge er _ to er_, etc.

(2) Result:

(1) Test set "n e w e r _" would be tokenized as a full word

(2) Test set "l o w e r _" would be two tokens: "low er_"

Properties of BPE tokens

Usually include frequent words

And frequent subwords

(1)Which are often morphemes like *-est* or *–er*

A morpheme is the smallest meaning-bearing unit of a language

(2)*unlikeliest* has 3 morphemes *un-*, *likely*, and *-est*

Word Normalization and other issues

Word Normalization

(1)Putting words/tokens in a standard format

(1)U.S.A. or USA

(2)uhhuh or uh-huh

(3)Fed or fed

(4)am, is, be, are

Case folding

(1) Applications like IR: reduce all letters to lower case

  (1) Since users tend to use lower case

  (2) Possible exception: upper case in mid-sentence?

    (1) e.g., ***General Motors***

    (2) ***Fed*** vs. ***fed***

    (3) ***SAIL*** vs. ***sail***

(2) For sentiment analysis, MT, Information extraction

  (1) Case is helpful (***US*** versus ***us*** is important)

Lemmatization

Represent all words as their lemma, their shared root

= dictionary headword form:

(1)am, are, is → be

(2)car, cars, car's, cars' → car

(3)Spanish quiero ('I want'), quieres ('you want')

→ querer 'want'

(1)He is reading detective stories

→ He be read detective story

Lemmatization is done by Morphological Parsing

(1)Morphemes:

(1)The small meaningful units that make up words

(2)Stems: The core meaning-bearing units

(3)Affixes: Parts that adhere to stems, often with grammatical functions

(2)Morphological Parsers:

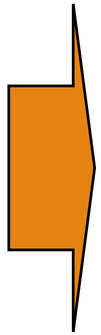
(1)Parse *cats* into two morphemes *cat* and *s*

(2)Parse Spanish *amaren* ('if in the future they would love') into morpheme *amar* 'to love', and the morphological features *3PL* and *future subjunctive*.

Stemming

Reduce terms to stems, chopping off affixes crudely

This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things-names and heights and soundings-with the single exception of the red crosses and the written notes.



Thi wa not the map we found in Billi Bone s chest but an accur copi complet in all thing name and height and sound with the singl except of the red cross and the written note
.

Dealing with complex morphology is necessary for many languages

(1) e.g., the Turkish word:

(2) Uygarlastiramadiklarimizdanmissinizcasina

(3) `(behaving) as if you are among those whom we could not civilize'

(4) Uygar `civilized' + las `become'

+ tir `cause' + ama `not able'

+ dik `past' + lar 'plural'

+ imiz 'p1pl' + dan 'abl'

+ mis 'past' + siniz '2pl' + casina 'as if'

Sentence Segmentation

!, ? mostly unambiguous but **period** "." is very ambiguous

(1) Sentence boundary

(2) Abbreviations like Inc. or Dr.

(3) Numbers like .02% or 4.3

Common algorithm: Tokenize first: use rules or ML to classify a period as either (a) part of the word or (b) a sentence-boundary.

(4) An abbreviation dictionary can help

Sentence segmentation can then often be done by rules based on this tokenization.

Summary